

SSAT Manual

R.J. Meeuws

rmeeuws@ce.et.tudelft.nl

G.Th. Aalbers

gtaalbers@ce.et.tudelft.nl

November 15, 2004

1 Introduction

The Delft University of Technology is implementing a tool chain for the modification of the SimpleScalar Tool Suite v3.0. This chain should contain tools for instructions, register files and caches. In this tool chain the SimpleScalar Architecture Tool (SSAT) is the tool for the addition and extension of register files in SimpleScalar v3.0. At this moment it works together with the SimpleScalar Instruction Tool (SSIT).

2 Usage

The SSAT is a command-line tool and has the following usage description:

```
Usage: ssat [options]
```

Options:

-a, --assembly	Path of the input assembly file
-c, --configuration	Path of the SSAT-configuration file
-h, --help	Show this help
-o, --output	Path of the output assembly file
-p, --parse	Only parse files, make no changes
-s, --simplerdir	Directory where the sim-outorder.c file resides
-v, --version	Show version information

Common Uses:

```
./ssat -c config.cfg -s /path/to/simplescalar
./ssat -c config.cfg -a /path/to/assembly.file
-o /path/to/processed.file
./ssat -c config.cfg -p
```

3 Tasks

The SSAT implements two basic tasks:

3.1 Modification of SimpleScalar source files

To be able to simulate new register files the SSAT modifies the SimpleScalar source code. This means that after modification the SimpleScalar Simulator must be re-compiled. At this time only the Out of Order simulator (sim-outorder) is supported, other simulators might be broken after modification. The following source files will be modified:

- sim-outorder.c
- regs.h
- machine.h
- machine.c

3.2 Register name substitution

The GNU assembler (GAS) port for SimpleScalar does not recognize register file extensions made by the SSAT, therefore this tool can substitute all real register names with register names that are recognized by the GAS. Furthermore it guarantees that the register names are compatible with the annotated instructions, i.e. it makes sure integer instructions use integer register names and floating point instructions use floating point register names. This functionality makes it possible to use different register names for register file extensions in your assembly file.

4 Files

ssat

This is the actual SSAT executable.

configuration-file

This file contains the configuration of the register file extensions. For a detailed description see below.

assembly-file

The assembly-file is the intermediate file that is obtained after application of the SSAT.

5 Configuration

Extensions of the register files of SimpleScalar v3.0 are specified in a configuration file. The configuration file consists of multiple directives. Each directive must reside on one line. All text behind a '#' symbol is assumed a comment. Comments are terminated by new lines. In the following section the possible directives are listed.

5.1 Directives

NEW_REGTYPE *[name], [size], [instances]*

This directive creates an entire new register file consisting of *[instances]* instances. The registers are named by appending the instance number to *[name]*, like "f0" is the first instance of the floating point registers (*[name]*='f'). The registers consist of *[size]* bytes, which means that registers can only consist of a multiple of 8 bits. The size must be a valid integer number and the instances must remain below 256 because of limitations of the GAS.

ALIAS_REGTYPE *[name], [alignment], [instances], [source name]*

This directive creates an alias to the register file with name *[source name]*, thus providing another way of accessing those registers. The alias is aligned to the source register file, i.e. each alias consists of *[alignment]* source registers. Given a valid alias instance A_i ($i \in \mathcal{N} | i \leq 256$) with alignment a ($a \in \mathcal{N} | a \leq 256$) consists of the source registers R_k ($k \in \mathcal{N} | k \in [ai, ai + a - 1]$). There cannot be more than $\frac{\max k}{a}$ instances of an alias.

EXTEND_REGTYPE *[name], [instances]*

This directive extends the existing register file with name *[name]* with *[instances]* instances. The resulting number of instances should not exceed 256. In case of extending the SimpleScalar integer register file, the special registers remain the same, so when we extend the 32 registers to 64, The special register \$31 (\$ra, return address) is not moved to \$63.

NEW_CTRL_REG *[name], [size]*

This directive extends the control register file. It adds exactly one register with *[name]* as name and consisting of *[size]* bytes. The total number of control registers cannot exceed 256, so in one modification of SimpleScalar one can add 253 new control registers (\$hi, \$lo and \$fcc are already taken).

5.2 Example

Example configuration (addition of mmx-like register-file)

```
# MMX register configuration
ALIAS_REGTYPE mmx, 1, 8, f # mmx register files
                          # aligned to float registers
```

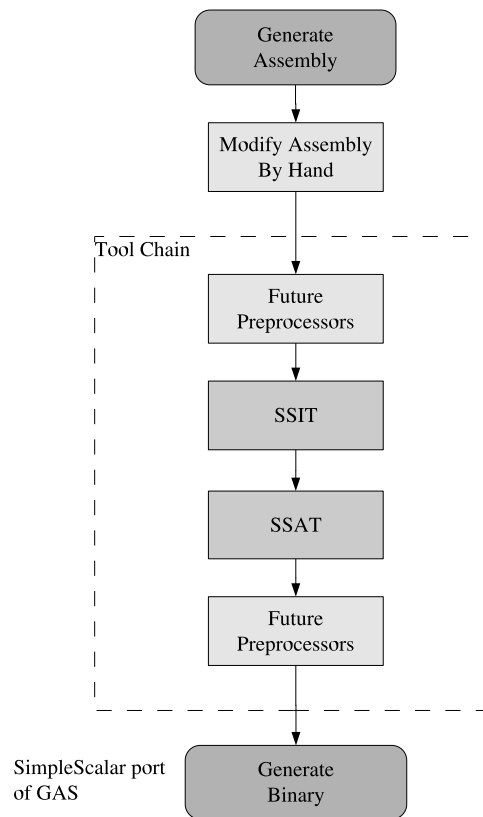


Figure 1: The location of SSAT in the tool chain.

```

NEW_CTRL_REG    cw, 2          # control word
NEW_CTRL_REG    sw, 2          # status word
NEW_CTRL_REG    tw, 2          # tag word
  
```

6 Integration with tool chain

6.1 Global integration

The SSAT was designed to function in a certain tool chain. This tool chain is still in development, but at the time SSAT was developed the SSIT was already finished. As one can see in figure 1, the SSAT performs its task after the SSIT has finished. This only applies for the transformation of assembly files; for these two tools the augmentation of the SimpleScalar simulator can be performed in any order.

6.2 Register Macros for SSIT

When adding new instructions to SimpleScalar with a SSIT configuration file one can use the following decoders, accessors and specifiers: (in the following XXX stands

for the register name in upper-case)

DSSAT_XXX, DSSAT_XXX(N)

This is the decoder macro. It should be used with *N* substituted with a specifier. For every new, aliased or control register such a decoder is defined. The control register decoders have no arguments.

SSAT_XXX, SSAT_XXX(N)

This is the accessor macro for reading the registers. The *N* argument again should be used with the *N* argument substituted with a specifier. The SSAT implements the registers in a way that the accessors return an array of bytes (**unsigned char[]**). The *N* argument does not apply to control registers,

SET_SSAT_XXX(N,EXPR)

This macro is the accessor for writing the registers. The *N* should be replaced with a specifier, except for control registers. *EXPR* should be an array of bytes (**unsigned char[]**) that has exactly the number of elements the register has.

SSAT_XXXS, SSAT_XXXD, SSAT_XXXT

These are the register specifiers for the *XXX* register file. The postfix *S*, *D* or *T* specifies if we want the source, destination or target *XXX* register.

7 Design

In this section we discuss the design of the SSAT. The tool consists of several c++-classes. The class diagram in figure 2 shows these classes and their relations. The design was made around the idea of performing the tasks in three phases: collect, validate and transform.

7.1 Collect

During the collect phase the configuration file and possibly an assembly file and SimpleScalar source files are read. Then every line of the configuration is parsed and for every directive an instance of its corresponding class (NewRegisterType, ExtendRegisterType, ControlRegisterType or AliasRegisterType) is created. If there are errors those are reported to the user and parsing will continue, but after the parsing phase the program will be terminated.

7.2 Validate

After the SSAT has made certain that the needed files are read correctly and do not contain any errors, the tool validates if the different register files in the new configuration and the existing configuration are compatible, i.e. do not have double names, do not exceed the maximum number of registers, do not alias non-existing

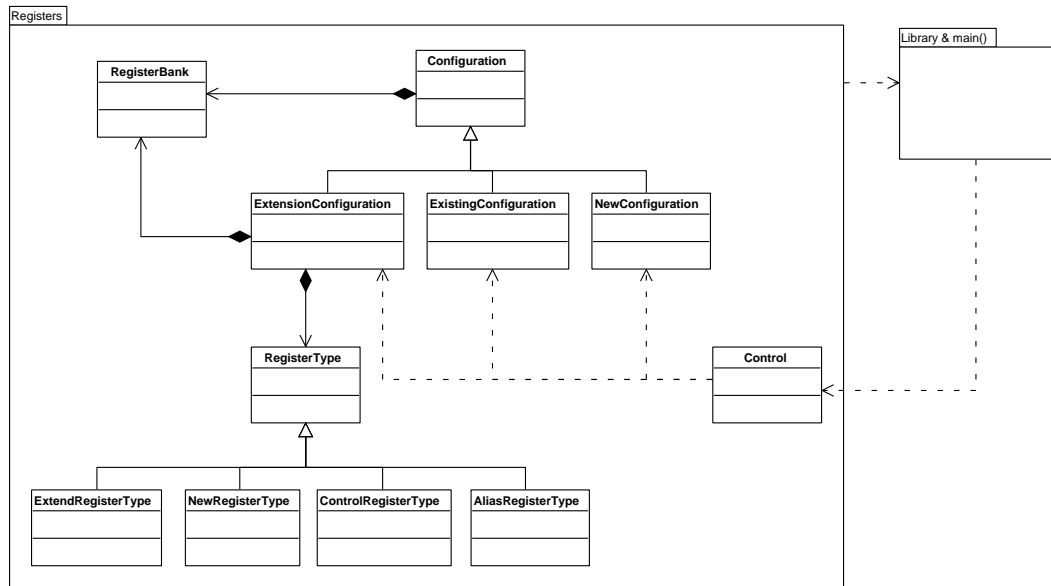


Figure 2: Class diagram of SSAT design.

registers, control registers or other aliases, do not extend aliases or control registers and are not misaligned to source registers. When errors are encountered, they are reported to the user and the SSAT will terminate when validation is complete. In case the SSAT was run with the '-p' option, the tool is finished.

7.3 Transform

When the SSAT has successfully validated the new configuration it will write the necessary files. For modification this means that the SimpleScalar files are modified and saved. In case of transformation the assembly-file is transformed and saved to the output-file. Errors may occur in this phase when the necessary files cannot be found, do not follow the expected format, are read-only or are already modified by the SSAT.

8 Diagnostics

The following error messages may occur during execution of the SSAT:

Collection errors

- Invalid configuration for DIRECTIVE on line X.
- Unknown configuration type encountered on line X: TOKEN.
- No configuration to apply on assembly file...

Validation Errors

- Extension of register NAME is not possible because it doesn't exist.
- New, Aliased or Control register name NAME clashes with simple-scalar native register.
- New, Aliased or Control register name NAME clashes with already added register.
- Extension of register file NAME beyond 256 instances not allowed.
- Extension of control register file beyond 256 instances not allowed.
- Source register file NAME of alias NAME2 cannot be found.
- Source register file NAME of alias NAME2 cannot be an alias itself.
- Aliased register NAME is not correctly aligned to NAME2.
- New register file NAME exceeds maximum number of 256 instances.

Transformation Errors

- Expected code signature CODE not found in FILE.
- Expected white line not found in FILE.